Automatic Translation from UML Specifications to B

Hung LEDANG

LORIA - Université Nancy 2 - UMR 7503

Campus scientifique - BP 239

54506 Vandœuvre-lès-Nancy Cedex - France

E-mail: ledang@loria.fr

Abstract

The translation from UML specifications to B specifications gives a way to use jointly UML and B in an unified, practical and rigorous software development. We can formally analyse UML specifications via their corresponding B formal specifications. This point is significant because B support tools are available. We can also use UML specifications as a tool for building B specifications, so the development of B specifications become easier.

So far, the rules for mapping data elements from UML specifications into B have been proposed. However, the problem of translating UML behavioral diagrams into B specifications has been an open issue. This point is the main concern in this paper. We are planning to propose derivation schemes to translate automatically UML behavioral diagrams into B specifications. Furthermore, the combination between the object refinement and the B refinement is also investigated.

Keywords: *UML, class operation, use case, event, B method, B abstract machine, B operation.*

1. Introduction

The Unified Modelling Language (UML)[23] has become a de-facto standard notation for describing analysis and design models of object-oriented software systems. The graphical description of models is easily accessible. Developers and their customers intuitively grasp the general structure of a model and thus have a good basis for discussing system requirements and their possible implementation. However, since the UML concepts have English-based informal semantics, it is difficult even impossible to design tools for verifying or analysing formally UML specifications. This point is considered as a serious drawback of UML-based techniques.

To remedy such a drawback, one approach is to develop the UML as a precise (i.e well defined) modelling language. The pUML[1] (precise UML) group has been created to achieve this goal. However the main challenge [6] of the pUML is to define a new formal formalism that has been up to now an open issue. Furthermore, the support tool for such a new formalism is perhaps another challenge.

In waiting for a precise version of UML and its support tool, the necessity to detect semantic defects inside UML specifications should be solved in a pragmatic approach (cf. [25, 5]) : formalising UML specifications by existing formal languages and then analysing UML specifications via the derived formal specifications. In this perspective, using the B language [1] to formalise/model UML specification has been considered as a promising approach [12, 24, 20, 22]. By formalising UML specifications in B, one can use B powerful support tools[2] like AtelierB [26], B-Toolkit [3] to analyse and detect semantic defects inside UML specifications (cf. [13]). On the other hand, we can also use UML specifications as a tool to develop B specifications which can be then refined automatically to the executable code [2, 10].

There are currently a set of derivation schemes to map into B all the concepts of static aspects of an UML specification such as class, attribute, association and inheritance. These derivation schemes are the PhD dissertations of Meyer[20] and Nguyen[22]. However, the problem of mapping UML behavioral diagrams into B has been an open issue. The work of Meyer and Nguyen considered only the state-chart diagrams, but they could not treat the transitions having multiple actions. Furthermore, the existing works coincide the concept class with the concept of an abstract machine, but indeed these two concepts do not coincide with each other. A class operation can affect the data from different classes but a B operation affects only data declared in the same abstract machine. For this reason, only basic class operations, which are local to classes, can be modelled. We cannot model class operations involving several classes. Consequently, with the current UML-B derivation schemes, we cannot translate interaction diagrams like sequence and collaboration to B.

[1] http:://www.cs.york.ac.uk/puml/

[2] This feature is one of the strong points of B over other formal languages like Z or VDM.

1.1. Objectives

Dealing with the modelling in B the UML behavioral diagrams is the main objective of the current work. We emphasise on the translation from use case, interaction and state-chart diagrams into B specifications. Furthermore, we study the possibility of formalising the UML refinement dependency by the B refinement. Finally, the support tool for automatic derivation from UML notations into B [20], is extended to take into account the new UML-B derivation schemes.

Section 2 gives a brief introduction to the B method. Section 3 presents our result and current work on the translation from UML behavioral diagrams to B specifications. In Section 4 we discuss the way to model the object refinement in B. Finally, some concluding remarks in Section 5 complete our presentation.

2. The B method

B [1] is a formal software development method that covers a software process from specification to implementation. The B notation is based on set theory, the language of generalised substitutions and first order logic. Specifications are composed of abstract machines that are similar to modules or classes. They consist of a set of variables, invariance properties relating to those variables and operations. The state of the system, i.e. the set of variable values, is only modifiable by operations which must preserve the invariant (Figure 1).

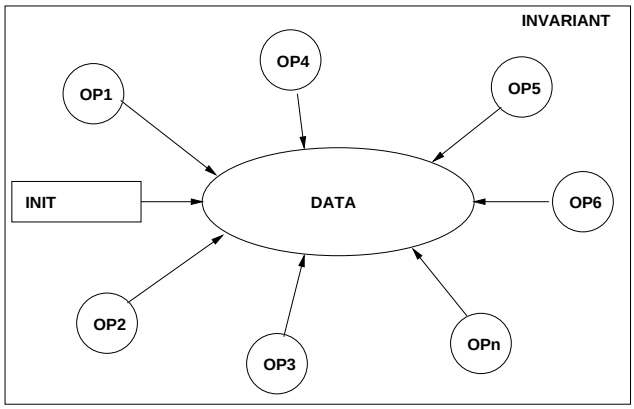

Figure 1. Intuitive view of a BAM

To express the post-conditions of operations, B provide the pseudo-code called generalised substitutions (GS). These GSs can be used to specify the non-determinism (at abstract specification level) and also the determinism (at implementation specification level). This point is a notable difference[3] with respect to Z and VDM, which use only logic expressions. These GSs provide a more familiar frame to specifiers by integrating the essential methodological aspects like invariant and refinement. Refinement can be seen as an implementation technique but also as a specification technique to progressively augment a specification with more details. At every stage of the specification, proof obligations ensure that operations preserve the system invariant. A set of proof obligations that is sufficient for correctness must be discharged when a refinement is postulated between two B components. Hence, by supporting proved refinement, B allows to go progressively from an abstract specification (non deterministic) to a deterministic specification that can be translated to a programming language (ADA, C and C++).

Another characteristic of the B method is that it have been designed to be automated easily. The generation of proof obligations (of the invariant preservation and of refinement correctness) obeys the simple rules that can be easily implemented in a piece of software. Furthermore, the support tools like AtelierB and B-Toolkit provide utilities to discharge automatically and interactively the generated proof obligations. Analysing the non-discharged proof obligations with the B support tools is a efficient and practical way to detect errors encountered during the specification development.

Finally, beside the refinement, B provide also structuring primitives like **INCLUDES**, **IMPORTS**, **USES** and **SEES** so that abstract machines can be composed in various ways. Thus, large systems can be specified in a modular way, possibly reusing parts of other specifications.

3. Automatic translation from UML behavioral diagrams to B specifications

3.1. From use case diagrams to B specifications

In [11] we have presented an approach for building B specifications from use-case models. Each use case is modelled as a B operation. To express in B the pre- and post-conditions of use cases, we propose modelling each use case and its involved classes in the same abstract machine[4]. For each use case having included use cases, its B operation is implemented by making invocations to B operations of the included use cases. A use case and its possible "extends" use cases are modelled as distinct B operations in the same abstract machine.

[3] This is seen as a strong point of B over Z and VDM.

[4] Otherwise, if we distribute data derived from different classes into different abstract machines, then by technical restriction of B, it is diffi cult even impossible to express in B the pre- and post conditions of use cases involving several classes.

By structuring use cases [7], we can arrange use cases in levels. The use cases at level one corresponds to "user-goal" use cases. The use cases, which are the included use cases of the ones at level one, are said at level two and so on. The bottom level of use cases is composed of basic operations of classes. It is to be noted that a use case can be at several levels. We derive for each use case level an abstract machine whose operations model use cases at that level and whose data are derived from all classes. The abstract machine for one level is implemented by importing the abstract machine in the next lower level (if any). The abstract machine in the bottom level are decomposed into abstract machines for classes and associations in the class diagrams.

3.2. From realization diagrams to B specifications

In [15, 14, 16] we have proposed a general approach for modelling all class operations. Like the proposals of Meyer and Nguyen, we model each class operation as a B operation. But our approach differs from theirs by proposing to group the class operation and its involved data in the same abstract machine. This point allows us to overcome the current shortcoming of modelling in B the pre-/postconditions of class operations involving several classes. In addition, we use the calling-called class operation dependency to arrange derived B operations into abstract machines. A calling-called pair relates a class operation - the calling operation - to one of its realization class operations - the called operation. We determine the calling-called class operation dependency from interaction diagrams and activity diagrams, which are often used to represent the realization of class operations (cf. chapters 19 and 27 in [4]). The B operation of the called operation participates in the implementation of the B operation which model the calling operation. That means : (i) the abstract machine for the called operation is imported in the implementation of the abstract machine for the calling operation and (ii) we use B implementation operation to model the realization of class operations.

Given a class diagram and realization (interaction or activity) diagrams for certain class operations. We have proposed a derivation procedure as followed :

1. establishing the calling-called class operation dependency from realization diagrams ;
2. if there is no circular calling-called class operation dependency[5], we are able to arrange class operations into layers (using two procedures : "division" and "dummy-promoting" [15]) such that :
   (a) there is no calling-called dependency among operations in the same layer ;
   (b) the basic operations, which do not have any called operation, are in the bottom layer ;
   (c) the system operations, which do not have any calling operation, are in the top layer ;
   (d) the operations in a layer differing from the bottom layer only have called operations in the next lower layer. For this purpose, certain operation is duplicated in several layers by using the "dummy-promoting" procedure.
3. after the allocation,
   (a) each layer gives rise to an abstract machine in which the B operations model the class operation in the associated layer[6] ;
   (b) an abstract machine that does not belong to the bottom layer, is implemented by importing the abstract machine for the next lower layer ;
   (c) the abstract machine for the bottom layer into abstract machines for classes and their associations (if any).

In the above approach for modelling class operation and calling-called dependency we did not treat asynchronous messages. We treat this problem in [17] (See also Section 3.3).

3.3. From state-chart diagrams to B specifications

The proposal of Meyer [21, 20] for modelling state-chart diagrams only works in cases without multiple actions related to a single transition. To overcome this shortcoming, we have proposed a two-stages approach [17] for modelling events :

1. modelling the effect of each event as a B abstract operation ;
2. implementing the B operation in the first step by calling B operations for the triggered transition and associated actions.

Given a set of classes and their state-chart diagrams, a integration procedure has been proposed to derive the corresponding B specification :

1. creating an abstract machine *System* to model all events in state-chart diagrams. The data in *System* are derived from classes and states ; this once again (cf. Sections 3.1 and 3.2) allows us to express easily the pre- and post specification of events ;

[5] This is still an open issue due to technical restrictions of the B language.

[6] Remember that, the data in each abstract machine are derived from the whole class diagram.

2. creating an abstract machine *Basic* to model all transitions, actions, state-checking and guard conditions ; the data in *Basic* are also derived from classes and states ;

3. decomposing *Basic* into abstract machines for classes and associations ;

4. implementing *System* by importing *Basic*.

To deal with asynchronous messages among state-chart diagrams, a B data structure modelling the signal type and an additional B operation modelling the sending of signal are supplemented for each type of signal. Details are also described in [17].

3.4. Further work : automatic generation of the B operations' content

Presently we can only automatically derive the architecture of B specifications from UML specifications. The data, the skeleton of B operations in the B specification are also automatically derived. In order to complete B specifications, we must fill up the body of B operations. For the purposes of a complete automation of transformation, we propose to attach to each element like use cases, events, actions, guard condition and class operations an OCL-based pre/-post specification. Hence, the abstract content of B operations can be derived by using OCL-B translation rules of Marcano [19]. The implementation content of B operations for use cases, events and non-basic class operations can be derived from corresponding diagrams (use case diagrams for use cases ; collaboration or activity diagrams for class operations ; state-chart diagrams for events). The precise rules will be proposed at a later stage.

4. Modelling the UML refinement dependency in B

Wills and D'Souza [9] have introduced four refinement types : operation refinement, model refinement, object refinement and action refinement in which the object refinement comprise all the action refinement, the model refinement and the operation refinement. The action refinement can be compared with the structuring of use cases, so the proposal for modelling use cases in Section 3.1 can be applied to model in B the action refinement. The approach for modelling class operations in Section 3.2 is totally appropriate for operation refinement. Section 4.1 describes an approach for modelling in B the model refinement.

4.1. Modelling in B the refinement constraints between two object models

Figure 2 presents an example of the refinement relation between two object design models. In this figure the package Package1 is refined by the package Package2. Both packages are supposed to contain an object-oriented specification of the same system or the same component at two abstract degrees. The refinement link is stipulated by a dependency relation between two packages. This relation is bounded with a constraint which expresses the gluing invariant of the refinement dependency.

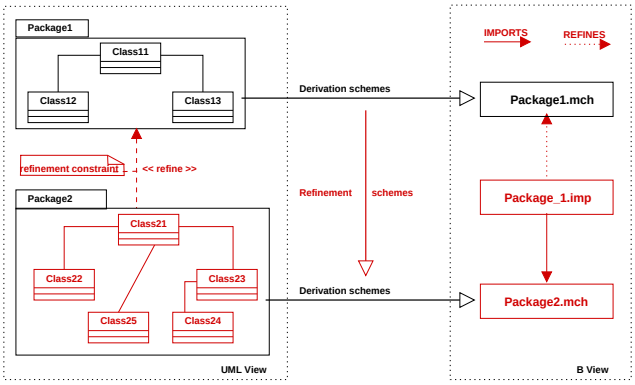

Figure 2. Graphic visualisation of refinement

We derive one B abstract machine for each package : the machine $Package1$ for the package Package1, the machine $Package2$ for the package Package2. Each machine only models class operations that are system operations (term used in [8]). There are two solutions for modelling the conformance of Package2 in comparison with Package1. The first solution (as showed in Figure 2) is to consider that the B abstract machine $Package2$ is used to implement the machine $Package1$. The refinement constraints between Package1 and Package2 are modelled as gluing invariant in the implementation of $Package1$. Another solution is to consider $Package2$ as a refinement of $Package1$. In this case we create $Package2$ as a refinement instead of an abstract machine. We prefer the first solution because it is more straightforward if we have several levels of abstraction ; in addition, it shares some points with modelling approaches for use cases and class operations in Sections 3.1 and 3.2

4.2. Further work : automatic generation of refinement constraints

For this purpose, we propose to use OCL expression to represent the refinements constraints between object mod-

els. Then we use OCL-B translation rules to map the OCL-based refinement constraints into B.

5. Conclusion

5.1. Evaluation methodology

We have validated the proposals by non trivial case studies. The work in [11] has been experimented with a case study on a controlling system for accessibility of buildings [18]. The approach for modelling interaction diagrams [15, 16] has been tried with several case studies : the pump component of a controlling system for petrol dispensing [8], the patterns like Client-Server, Broker. The approach for modelling state-chart diagrams has been applied for modelling a lift system. Currently, we are trying apply the proposal in Section 4 with the pattern specialisation.

5.2. Concluding remarks

The results in [11, 16, 14, 15, 17] provide a complete framework for deriving B specifications from UML structure and behavioral diagrams. Hence, the conformance between two aspects (the structure and the behaviour) of UML specifications can be formally verified by analysing the corresponding B specification. In [13], we have envisaged the following verifications on UML specifications : `(i)` the consistency of the class invariant ; `(ii)` the conformity of object and state-chart diagrams regarding the class diagrams ; `(iii)` the conformity of class operations, use cases regarding the class invariant ; `(iv)` the class operation calling-called dependency and `(v)` the use case structuring.

References

[1] J.R. Abrial. *The B Book - Assigning Programs to Meanings*. Cambridge University Press, 1996. ISBN 0-521-49619-5.

[2] A. Amelot and D. Dollé. Le raffinement automatique. Available at http://www3.inrets.fr/B@INRETS/Events/2001-ESTAS/actes/MTI-2001-ESTAS.*, 2001. Slides.

[3] B-Core(UK) Ltd, Oxford (UK). *B-Toolkit User's Manual*, 1996. Release 3.2.

[4] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unifi ed Modeling Language User Guide*. Addison-Wesley, 1998. ISBN 0-201-57168-4.

[5] J.M. Bruel. Integrating Formal and Informal Specification Techniques. Why? How? In *the 2nd IEEE Workshop on Industrial-Strength Formal Specifi cation Techniques*, pages 50–57, Boca Raton, Florida (USA), 1998. Available at http://www.univ-pau.fr/∼bruel/publications.html.

[6] J.M. Bruel, J. Lilius, A. Moreira, and R.B. France. Defining Precise Semantics for UML. In *Object-Oriented Technology*, LNCS 1964, pages 113–122, Sophia Antipolis and Cannes (F), June 12-16, 2000. ECOOP 2000 Workshop Reader.

[7] A. Cockburn. Structuring Use Cases with Goals. http://members.aol.com/acockburn/papers/usecases.htm, 1997.

[8] D. Coleman, P. Arnold, St. Bodoff, Ch. Dollin, H. Gilchrist, F. Hayes, and P. Jeremaes. *Object-Oriented Development : The Fusion Method*. Prentice Hall, 1994.

[9] D. F. D'Souza and A. C. Wills. *OBJECTS, COMPONENTS AND FRAMEWORKS WITH UML : The Catalysis Approach*. Addison-Wesley, 1998.

[10] R. Laleau and A. Mammar. A Generic Process to Refine a B Specification into a Relational Database Implementation. In *ZB 2000: Formal Specifi cation and Development in Z and B*, LNCS 1878, York (UK), August/September 2000. Springer.

[11] H. Ledang. Des cas d'utilisation à une spécification B. In *Journées AFADL'2001 : Approches Formelles dans l'Assistance au Développement de Logiciels*, Nancy (F), 11-13 juin, 2001. http://www.loria.fr/∼ledang/publications/afadl01.ps.gz.

[12] H. Ledang. Formal Techniques in the Object-Oriented Development: an Approach based on the B method. *PhDOOS2001: the 11th ECOOP Workshop for PhD Student in Object-Oriented Systems*, Budapest (Hu), http://www.st.informatik.tu-darmstadt.de/phdws/wstimetable.html, June 18-19, 2001. http://www.loria.fr/∼ledang/publications/PhDOOS01.ps.gz.

[13] H. Ledang and J. Souquières. Formalizing UML Behavioral Diagrams with B. In *the Tenth OOPSLA Workshop on Behavioral Semantics: Back to Basics*, Tampa Bay, Florida (USA), October 15, 2001. http://www.loria.fr/∼ledang/publications/oopsla01.ps.gz.

[14] H. Ledang and J. Souquières. Integrating UML and B Specification Techniques. In *the Informatik2001 Workshop on Integrating Diagrammatic and Formal Specifi cation Techniques*, Vienna (Autria), September 26, 2001. http://www.loria.fr/∼ledang/publications/informatik01.ps.gz.

[15] H. Ledang and J. Souquières. Modeling class operations in B : a case study on the pump component. Technical Report A01-R-011, Laboratoire Lorrain de Recherche en Informatique et ses Applications, March 2001. http://www.loria.fr/∼ledang/publications/UML01.ps.gz.

[16] H. Ledang and J. Souquières. Modeling Class Operations in B: Application to UML Behaviral Diagrams. In *ASE2001: the 16th IEEE International Conference on Automated Software Engineering*, Loews Coronado Bay, San Diego (USA), November 26-29, 2001. http://www.loria.fr/∼ledang/publications/ase01.ps.gz.

[17] H. Ledang and J. Souquières. New Approach for Modeling State-Chart Diagrams in B. Technical Report A01-R-082, Laboratoire Lorrain de Recherche en Informatique et ses Applications, September 2001. http://www.loria.fr/∼ledang/publications/state-chart-modeling.ps.gz.

[18] Y. Ledru, G. Padiou, and J. Jaray. Étude de cas: Système de contrôle d'accès. http://www-lsr.imag.fr/afadl2000/EtudeDeCas/, 2000.

[19] R. Marcano and N. Lévy. Transformation d'annotations OCL en expressions B. In *Journées AFADL'2001 : Approches Formelles dans l'Assistance au Développement de Logiciels*, Nancy (F), 11-13 juin, 2001.

[20] E. Meyer. *Développements formels par objets: utilisation conjointe de B et d'UML*. PhD thesis, LORIA - Université Nancy 2, Nancy (F), mars 2001.

[21] E. Meyer and J. Souquières. A systematic approach to transform OMT diagrams to a B specification. In *FM'99 : World Congress on Formal Methods in the Development of Computing Systems*, LNCS 1708, Toulouse (F), September 1999. Springer-Verlag.

[22] H.P. Nguyen. *Dérivation de spécifications formelles B à partir de spécifications semi-formelles*. PhD thesis, Conservatoire National des Arts et Métiers - CEDRIC, Paris (F), décembre 1998.

[23] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1998. ISBN 0-201-30998-X.

[24] C. Snook and M. Butler. Verifying Dynamic Properties of UML Models by Translation to the B Language and Toolkit. Technical Report DSSE-TR-2000-12, Declarative Systems & Software Engineering Group, Department of Electronics and Computer Science University of Southampton, September 2000. Available at http://www.dsse.ecs.soton.ac.uk/techreports/2000-12.html.

[25] C. Snook and R. Harrison. Practitioners Views on the Use of Formal Methods: An Industrial Survey by Structured Interview. *Information and Software Technology*, 43:275–283, March 2001.

[26] STERIA - Technologies de l'Information, Aix-en-Provence (F). *Atelier B, Manuel Utilisateur*, 1998. Version 3.5.